Universidad Autónoma de Madrid
Escuela Politécnica Superior


Grado en Ingenería Informática

Trabajo Fin de Grado

# INTEGRACIÓN DE LA HERRAMIENTA SKYTUNEUP CON LA PLATAFORMA AZURE

Ángel Manuel Martín Canto
Jesús Gironda
Eloy Anguiano Rey

Julio 2018

# INTEGRACIÓN DE LA HERRAMIENTA SKYTUNEUP CON LA PLATAFORMA AZURE

Ángel Manuel Martín Canto
Jesús Gironda
Eloy Anguiano Rey

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio 2018

# Abstract

The design and implementation of a software project to expand the functionality of Skytuneup is given. Skytuneup is a SaaS, or Software as a Service, tool that focuses on managing groups of virtual machines on public clouds. The remarkable part about this management is the mixed use of on-demand paid instance with instances on reduced pricing(spot price, low priority) to allow for savings without sacrificing availability. This tool also offers scheduled scaling and backup management. The focus of this thesis is the integration of this with Microsoft Azure. Beforehand the tool only worked with Amazon Web Services. The actual context of payment models on the public cloud is discussed. The workings and arquitectural design of the tool is explained. Also explained are the desing choices relative to the new development and their effect of the implementation. The tests that the tool has passed are describen and, after concluding, hints about future work are given.



# Keywords

cloud, spot, SaaS, IaaS, Skytuneup, Azure, platform, development

# Resumen

Se describe el diseño e implementación de un proyecto de software para ampliar la funcionalidad de Skytuneup. Skytuneup es una herramienta SaaS o software como servicio que se dedica a gestionar grupos de máquinas virtuales en nubes publicas. Esta gestión destaca combinar instancias de pago bajo demanda con instancias de pago reducido(precio flotante, baja prioridad) para permitir ahorro sin perder disponibilidad. La herramienta también ofrece escalado programado y gestión de backups. El foco de este trabajo es la integración de esta herramienta con Microsoft Azure. Anteriormente la herramienta solo funcionaba con Amazon Web Services. Se discute el contexto actual de los modelos de pago en nube pública. El funcionamiento y diseño arquitectural de la herramienta es explicado. También se explican las decisiones de diseño propias del nuevo desarrollo y como han afectado a la implementación. Se describen las pruebas que la implementación ha superado y, después de concluir, se dan pistas sobre el trabajo futuro.



# Palabras clave

nube, spot, SaaS, IaaS, Skytuneup, Azure, plataforma, desarrollo

# Agradecimientos

A mi madre por preocuparse mucho por mi TFG.

A mi tutor y CTO de Enimbos, Jesús Gironda, por darme la oportunidad de trabajar en un proyecto como este.

# Índice general

# Índice de Figuras

# Capítulo 1

# Introducción

En este trabajo se presenta un desarrollo sobre una plataforma para la gestión y el ahorro de costes en entornos de nube pública. En la siguiente sección se describen los motivos detrás de la realización del trabajo y que objetivos se persiguen.

## 1.1. Motivación

La nube es hoy en día una parte muy importante del tejido económico digital. Muchas empresas y otros organismos han incorporado la nube en su estrategia ya sea creando proyectos en la nube, migrando proyectos a la nube o incorporando la nube como una parte de su infraestructura preexistente. Esta presencia está creciendo y las predicciones apuntan a que la nube será una parte integral para la mayor parte de las compañías tecnológicas. Según la prestigiosa firma de consultoría IT Gartner[1], para 2019 las nuevas inversiones de al menos 30 de las 100 casas de software más grandes tendrán por entorno destino únicamente la nube, y para 2020 la nube venderá más poder computacional que el existente en el resto de CPDs del mundo. También aseguran que se trata de un crecimiento sólido, en su modelo hype curve[2] ubican a la nube en la pendiente iluminada, lejos de crecimientos de burbuja.

La nube muestra tanto crecimiento por las nuevas oportunidades que ofrece frente a la infraestructura IT tradicional. Pero la nube no es una mejor versión de los actuales paradigmas sino un conjunto de nuevas ideas y tecnologías que ofrecen ventajas a aquellos que se adapten mejor y puedan cambiar su forma de trabajar para apoyarse en los puntos fuertes del nuevo paradigma. Algunas de las ideas más en sintonía con la nube son viejas, como el balanceo de carga y la replicación distribuida de la información, pero otras ideas son nuevas, han nacido con la nube o han recibido un enorme empujón desde la relativa semioscuridad a la practica común. La infraestructura como código, el uso extendido de componentes sin estado persistente y el aprovisionamiento dinámico de la capacidad son ejemplos de este segundo grupo de ideas.

Cuando se trata la nube desde un enfoque financiero y empresarial se señala como clave que la nube supone sustituir inversión por gasto corriente. También se compara la nube con servicios tales como el agua, la electricidad y el gas en el sentido de que permite disponer de computación bajo demanda y en un formato que permite casi cualquier uso posible. Pero hacer esta visión realidad en toda su extensión tiene prerequisitos técnicos que acaban llamando al uso de las ideas comentadas anteriormente. Por ejemplo, si muchos de los componentes de un sistema no tienen estado uno puede tratar su almacenamiento y computación como transitorio y redimensionable. Siendo concretos, una aplicación web

en un entorno de nube debería guardar todo el estado relevante en una base de datos en lugar de almacenar total o parcialmente datos de negocio en su propio disco, idealmente incluso la información de sesión debería guardarse en un componente separado al servidor que ejecuta la aplicación.

El enfoque que compara la nube con un enchufe que suministra computación tiene, al igual que los enchufes reales, más complejidad de la que al principio es aparente a la vista. Nuestras líneas eléctricas tienen limites, tarifas diferentes según la hora, diferentes niveles de servicio garantizados. Incluso algunos grandes consumidores industriales están dispuestos a aceptar interrupciones a cambio de compensación o descuentos. La nube avanza, por el interés combinado de proveedores y consumidores, a modelos de pago más sofisticados y con más opciones.

Dada la conexión entre el aspecto financiero y las decisiones técnicas en la nube esta evolución de los modelos de pago supone nuevas oportunidades que requieren de soluciones para aprovecharlas plenamente.

Esta necesidad, combinada con el creciente papel de la nube, otorga una gran importancia a herramientas como Skytuneup[1] de Enimbos[2]. Este trabajo se centra en una nueva integración de la herramienta Skytuneup con la plataforma de nube pública Azure[3] de Microsoft.

## 1.2. Objetivos

El objetivo de este trabajo es integrar Skytuneup con la plataforma de nube pública Microsoft Azure.

La integración debe adecuarse en la medida de la posible a la integración existente con Amazon Web Services[4]. Esto es para que Skytuneup facilite el traslado de competencias en la gestión de entornos en nubes distintas y para perseguir una funcionalidad generalizable y multi-nube en vez de una anquilosada en en los detalles de un proveedor particular. La integración con Azure debe por tanto replicar la funcionalidad existente para AWS en la medida de lo posible. Concretamente debe ser posible gestionar grupos de máquinas virtuales en distintas modalidades de pago, programar acciones de escalado de capacidad y realizar backups de forma automática.

El desarrollo debe ser fácil de mantener y de probar. Se perseguirá una estrategia de pruebas fin a fin que se centren en funcionalidades completas sin perjuicio de probar llamadas individuales de la API y de realizar pruebas de unidad donde se consideren valiosas. La herramienta podría ser utilizada en régimen de beta privada sobre entornos reales en el futuro próximo por lo que la estabilidad y correcto funcionamiento apremian.

## 1.3. Organización de la memoria

Esta memoria está organizada de la forma que se describe a continuación:

- El Capítulo 2 trata el estado del arte. En primer lugar se dedica un espacio al contexto en el que el trabajo se ubica, esto es, la nube pública y sus distintos modelos

[1] `http://skytuneup.com/`
[2] `https://www.enimbos.com/`
[3] `https://azure.microsoft.com/`
[4] `https://aws.amazon.com/`

de pago. Después de analizan las soluciones existentes incluyendo herramientas de cada proveedor de nube y herramientas de terceros con o sin capacidad multi-nube.

- En el Capítulo 3 se explica el diseño de la nueva integración. Primero explicando la metodología con la que se han tomado las decisiones, después dando cuenta de requisitos de alto nivel y del diseño previo de Skytuneup para, finalmente, tratar las decisiones adoptadas y algunos detalles especificos de su implementación.
- En el Capítulo 4 da cuenta de las pruebas a las que se ha sometido el sistema y se exponen los resultados del esfuerzo de implementación.
- Finalmente el Capítulo 5 contiene las conclusiones de la memoria y habla del trabajo futuro en Skytuneup incluyendo cambios planeados, mejoras que se derivan naturalmente de las conclusiones y direcciones hipotéticas que se podrían adoptar.

# Capítulo 2

# Estado del arte

## 2.1. Contexto del trabajo

Para dar una visión inteligible del estado del arte primero se va a exponer el contexto en el se ubican los esfuerzos de herramientas como la que ocupa este trabajo. Primero se dará una visión de la computación en la nube y de los principales proveedores de nube pública. Después se trataran los distintos modelos de pago existente, esto derivará en una exposición de la problemática seguida de formas en las que se puede abordar.

### 2.1.1. Computación en la nube

La nube es un paradigma dentro de las tecnologías de la información que se basa en el uso de un grupo de recursos de computación, almacenamiento y transferencia de datos para varios fines. Esta multiplicidad de fines se combina con una separación lógica y segura de los distintos proyectos que usan los recursos de forma que proyectos de actores distintos(compañías, individuos, departamentos) no interfieran unos con otros aún cuando utilicen los mismos recursos físicos.

Los servicios de la nube pueden tener varios niveles de abstracción con respecto al origen último de la computación, el almacenamiento o la red. Por ejemplo, una nube puede ofertar al mismo tiempo un almacenamiento por bloques que puede presentarse como un disco duro a una máquina virtual y un servicio de almacenamiento en red con muchas opciones configurables al que se puede accederse desde muchas máquinas virtuales.

El paradigma opuesto a la nube es la infraestructura sobre premisas, esto es, sistemas organizados para un propósito concreto. Por ejemplo, un ordenador cuyo propósito es servir una pagina web conectado a otro cuyo propósito es ser una base de datos usando un switch cuyo propósito es unicamente conectar ambas máquinas y una puerta de enlace al exterior. En cambio, en la nube, los recurso son mas fungibles, se pueden crear y destruir bajo demanda, permiten el pago por uso y se pueden distribuir y replicar globalmente con mayor facilidad.

Se distinguen varios tipos de nube según su acceso, la nube pública es propiedad de un proveedor de servicios de nube y está disponible a varias organizaciones cliente, la nube privada es propiedad de una organización y solo se usa para propósitos de la organización, y la nube híbrida es una combinación de un entorno en nube privada con un entorno en nube pública.

El presente trabajo se centra en nube pública.

### 2.1.2. Grandes proveedores de nube pública

La ya mencionada firma de consultoría IT Gartner evalúa todos los años a los proveedores de infraestructura en nube y los puntúa en ejecución y visión, aquellos que sobresalen en ambas aéreas son catalogados como lideres del sector. En el último informe[4] los lideres fueron Amazon, Microsoft y Google.


Figura 2.1: Cuadrante mágico para IaaS Cloud 2017 de Gartner

### 2.1.3. Modelos de pago en la nube

En la nube pública existen varios modelos de pago. A continuación describimos aquellos más extendidos explicando los detalles de su implementación en cada uno de los proveedores de nube más importantes.

#### Pago bajo demanda

El más sencillo. Consiste en el pago de un cantidad de determinada por el proveedor a cambios de una cantidad de recursos por el tiempo en que se usen. Garantiza la disponibilidad de los recursos una vez aprovisionados y solo se realiza mientras se usen los recursos pero también es el modelo de pago más oneroso. Todos los proveedores usan este modelo de pago como opción principal y por defecto para la mayor parte de recursos.

#### Pago adelantado y pago de larga duración

Si uno paga por adelantado el uso de un recurso por un periodo extenso se pueden aplicar descuentos substanciales. Este es el modelo de pago adelantado o de recursos reservados. Una vez hecho el pago, estos de recursos de pueden usar sin coste adicional

por el periodo en cuestión, se usen o no efectivamente los recursos no se produce la devolución del pago. Este es el caso en AWS y Azure.

Una variación del modelo, utilizada en Google Cloud, consiste en aplicar automáticamente descuentos por el uso continuado de un recurso.

En este modelo los proveedores de nube se benefician de una demanda más predecible y de un flujo de ingresos menos volátil y los clientes de un precio reducido.

#### Pago spot y pago de baja prioridad

Los proveedores de nube se enfrentan a una demanda fluctuante que han de suplir con una infraestructura finita que no puede crecer tan elásticamente. Para ser capaces de afrontar los picos de demanda los proveedores dimensionan su infraestructura por encima de la utilización media. La mayor parte del tiempo el resultado son recursos físicos ociosos. Estos recursos ociosos, a pesar de entrar en modo de bajo consumo, siguen consumiendo electricidad, generando calor residual, degradándose y devaluándose sin producir beneficio económico al proveedor de nube que mantiene el CPD.

Para rentabilizar esta capacidad los proveedores la ofrecen en modelos de pago reducido. Hay dos subtipos de este modelo. Uno es el precio spot o flotante, donde la capacidad se subasta en mercado y los clientes revelan cuanto están dispuestos a pagar por un recurso, si el precio de mercado está por debajo se les cobra a precio de mercado, si está por encima se les retiran los recursos. Esto es lo que se ofrece en Amazon Web Services para recursos de computación en forma de Spot Instances. La otra modalidad son los recursos de baja prioridad, en este caso el precio es fijo pero muy inferior al precio habitual, habitualmente un 20 % del primero. En caso de que el proveedor reciba una demanda por encima de su capacidad primero retirara estos recursos de baja prioridad. Esta modalidad es la que encontramos en Google Cloud Platform bajo el nombre de Preemptible Instances y en Microsoft Azure bajo el nombre de Low Priority VMs.

#### Acuerdo específico

Adicionalmente, aunque se encuentra fuera del ámbito de este trabajo y es una opción no estándar por naturaleza, existe la posibilidad de que grandes clientes forjen un acuerdo especifico con un proveedor de nube que incluye un criterio de pago especial.

### 2.1.4. Problemas del modelo de pago spot o de baja prioridad

El modelo de pago reducido, a pesar de ser más económico, tiene la principal desventaja para el cliente de nube de que el recurso puede dejar de ser utilizable en cualquier momento con muy poco o nulo aviso.

Hay tareas de procesamiento que no se ven muy perjudicadas por esta restricción por ser reanudables o por no tener requerimientos duros de duración. Generalmente tareas de procesamiento en bloque como por ejemplo análisis en diferido en grandes series de datos, aplicaciones de aprendizaje automático, procesamiento de video, etc.

Por otro lado, muchas aplicaciones tienen requerimientos de disponibilidad continua y de latencia baja. Quizás el caso más característico es un servicio web expuesto al público. Por lo tanto el problema es que estas cargas de trabajo parecen tener vedado el uso de recursos en pago reducido.

Exploremos un escenario común. Supongamos que tenemos una aplicación que puede soportar la caída de una pequeña parte de sus servidores sin causar fallos de servicio, si

uno intentase usar exclusivamente máquinas virtuales de pago reducido se producirían fallos de servicio. A pesar de que la aplicación soporte la desaparición de uno cualquiera de los servidores seguramente no podría seguir prestando servicio si desapareciesen todos o la mayor parte simultáneamente. Y justamente ese es un escenario muy plausible, los recursos de pago reducido solo son desaprovisionados en situaciones de alta demanda pero si un solo recurso se desaprovisiona eso ya señala una alta demanda y, por tanto, una mayor probabilidad de que otros recursos le sigan.

Parece que la problemática no es puede evitar simplemente con un diseño arquitectural resiliente. Sin embargo, para aplicación diseñada de esta forma puede existir una solución que permita el uso de recursos en pago reducido.

### 2.1.5. Combinar recursos en pago reducido con recursos en pago bajo demanda como solución al problema

Una solución para asegurar la disponibilidad en esos escenarios es mantener un numero de recursos en modelo de pago por uso normal. Si la proporción entre capacidad en la modalidad normal y capacidad en la modalidad de pago reducido es la apropiada es posible seguir prestando servicio si la capacidad de pago reducido se pierde. Por supuesto, el servicio puede ver su rendimiento degradado, por ejemplo, un servicio web podría seguir atendiendo las peticiones pero las colas de peticiones por responder se alargarían y la latencia subiría. Este golpe en el rendimiento se puede aliviar cambiando la proporción a cambio de un menor ahorro.

Hay aplicaciones en las que los perjuicios económicos de una perdida de servicio temporal no son lo suficientemente importantes como para gastar recursos en protegerse perfectamente. En estos casos es posible asumir un riesgo de perdida de servicio ajustable variando la proporción.

### 2.1.6. Diversificación y arbitraje entre clases de máquinas

Otra solución que puede ayudar a mitigar el problema enormemente en primer lugar y que se puede aplicar en combinación con la anterior es diversificar los recursos de pago reducido entre diferentes clases similares de recurso.

Los proveedores de nubes ofrecen tamaños predefinidos de máquinas virtuales, incluidas aquellas de pago reducido. Es posible lograr un balance más ventajoso entre disponibilidad y rendimiento usando máquinas de varios tamaños. En el modelo de pago con precio flotante las subastas son especificas de cada clase de máquina y en ocasiones el precio de una clase de máquinas puede subir más rápido que el conjunto, usar una flota de máquinas protege de la posibilidad de que una subida aislada produzca la retirada de muchas máquinas, si añadimos el cambiar el tipo de maquinas podemos volver rápidamente al estado deseado e incluso ahorrar dinero. En el modelo de pago reducido pero fijo no se obtiene un ahorro económico directo por reasignar las maquinas en clases dinámicamente pero se sigue consiguiendo más protección contra la terminación de las instancias y eso, por la via de permitir una mayor proporción de recursos en pago reducido con la misma disponibilidad ayuda a conseguir ahorros indirectamente.

Adicionalmente existe otro eje sobre el cual se pueden distribuir recursos y realizar un arbitraje para ahorrar costes, las zonas de disponibilidad del proveedor de nube. Es lo habitual que un proveedor de nube ofrece recursos en distintas ubicaciones geográficas pero además también es común que una misma ubicación geográfica cuente con varios

centros de procesamiento de datos separados o que no compartan ninguna infraestructura crítica. Los proveedores exponen esta realidad mediante las zonas de disponibilidad, cada recurso en una ubicación geográfica tiene una asociada y si un sistema en la nube cuenta con componentes redundantes en cada zona puede estar mucho más protegido ante caídas o degradaciones de servicio en un zona individual. Por otro lado, dado que las zona no comparten infraestructura los recursos libres en pago reducido pueden encontrarse en distinto grado de abundancia, esto se manifiesta en el modelo de precio flotante con precios distintos por cada zona. Dado que en la mayor parte de las ocasiones el cliente es indiferente de la zona o reparte sus recursos en todas las zonas el proveedor es libre de asignar la capacidad de las zonas de formas balanceada y las oportunidades de arbitraje son menores. No obstante distribuir los recursos en distintas zonas sigue siendo una practica beneficiosa y recomendada por lo que no se debería olvidar al usar recursos de pago reducido.

### 2.1.7. Enimbos Skytuneup

La herramienta Skytuneup de Enimbos implementa las anteriores soluciones. A continuación se explica con más detalle que funcionalidad aporta Skytuneup. Antes de este trabajo Skytuneup solo operaba con AWS, este es el estado que aquí se detalla, los cambios introducidos en este trabajo se tratan más adelante.

Skytuneup es una herramienta en la categoría de SaaS(Software as a Service), esto es, software como servicio. El usuario de Skytuneup no tiene que instalar ningún software, puede interactuar con Skytuneup a través del portal web de la herramienta. Inicialmente el usuario ha de autorizar a Skytuneup en su cuenta de AWS, solo es necesario un subconjunto reducido de permisos y Skytuneup no tiene acceso a la información procesada por los sistemas del usuario en ningún momento.

#### Grupos Skyspot

La funcionalidad más destacable de Skytuneup son los grupos Skyspot. Estos tienen las siguientes características:

- **Reducción de costes**
  Al usar instancias de precio reducido es posible bajar los costes hasta un 90 %.
- **Alta disponibilidad**
  Dando la opción de mantener maquinas en la modalidad de pago bajo demanda es posible asegurar unos niveles determinados de servicio.
- **Diversificación**
  Skytuneup distribuye los recursos entre clases de recurso y zonas de disponibilidad.
- **Asignación dinámica e inteligente de recursos**
  Skytuneup recalcula el precio de puja necesario para equilibrar ahorro y disponibilidad.
- **Ahorro-disponibilidad configurable**
  El balance deseado es configurable. Este parámetro se propaga y afecta la proporción de de recursos en las distintas modalidades de pago, en como de agresivamente puja la subasta iterativa por los recursos y en como persigue o no el máximo ahorro frente a la diversificación en tipos y zonas.

- **Escalado automático**
  Los grupos skyspot ofrecen el que es uno de los pilares de la computación en la nube, el escalado automático. Los grupos se pueden configurar para cambiar de tamaño creciendo o decreciendo si un parámetro, como el uso de CPU cruza determinados umbrales. El parámetro se puede elegir entre cualquier integrado en AWS o se pueden usar métricas arbitrarias de usuario registradas en AWS.

- **Escalado manual**
  Adicionalmente se puede optar por establecer las dimensiones en equilibrio del grupo manualmente.

#### Tuneups

Skytuneup ofrece Tuneups, estos son acciones programadas que permiten escalar horizontal como verticalmente grupos Skyspot o grupos de autoescalado tradicionales de AWS a cualquier hora del día. Esta funcionalidad permite aprovechar la flexibilidad de la nube y es ideal para entornos que solo tienen que prestar servicio en horas lectivas o que se encienden bajo demanda pero se han de apagar automáticamente para evitar gastos.

#### Backups

Skytuneup también provee de una solución de backup de instancias y bases de datos. Los backups son configurables en frecuencia y política de retención. Esta funcionalidad automáticamente y facilita el proceso de backup y se basa en las primitivas para guardar el estado de instancias y discos del proveedor de nube. Estas primitivas guardan el estado de los discos a nivel de bloque en vez de realizar un backup lógico por lo que son compatibles con cualquier base de datos o software y tienen equivalentes similares en otros proveedores de nube.

## 2.2. Soluciones similares

Ahora pasamos a la exploración del estado del arte más allá de Skytuneup. A continuación se describen las opciones existentes que proveen funcionalidad similar a Skytuneup. Dado que Skytuneup combina muchas funcionalidades ninguna opción tiene un rango de funcionalidad idéntico y muchas son herramientas o técnicas con un objetivo más especializado que cubren una faceta de Skytuneup. No obstante se ha centrado el análisis en la funcionalidad central de los grupos mixtos con diferentes modalidades de pago y tipos de instancia.

El estado de arte evoluciona rápidamente y esta sección ha sido revisada varias veces durante la realización. Naturalmente algunos de las soluciones más reciente de las que se exponen han llegado mientras en una fase tardía de diseño o implementación avanzada del presente trabajo por lo que no han podido informar mucho las decisiones tomadas. Además, el inicio del proyecto Skytuneup precede a la aparición de gran parte de los casos presentados ya que se trata de una solución pionera. La propia realización de este trabajo convierte a Skytuneup en la primera herramienta para aprovechamiento de recursos de pago reducido en abarcar varias nubes.

### 2.2.1. Soluciones de proveedor

En esta sección se tratan soluciones de los propios proveedores de nube. Estas soluciones generalmente se circunscriben a la nube del propio proveedor.

#### AWS EC2 Spot Fleet

AWS EC2 Spot Fleet[5] es un servicio en AWS que existe desde 2015 y permite manejar un grupo de instancias de precio flotante o spot. Una spot fleet se define indicando los tipos de máquinas spot a usar acompañadas de un valor numérico que debe representar cuanto rendimiento proveen a la aplicación y dando un valor númerico como objetivo de rendimiento para toda la flota. A partir de ese momento AWS intentan alcanzar la capacidad deseada combinando instancias, la forma en que lo hace es configurable pudiendo elegir entre una aproximación que minimice el coste y otra que diversifique entre todos los tipos de instancias. Las estrategias mixtas como las de Skytuneup no son una opción.

En 2016 se añadieron capacidad de autoescalado a Spot Fleet[6].

#### AWS EC2 Fleet

AWS EC2 Fleet[7] es una generalización de EC2 Spot Fleet que no se limita a instancias spot sino que permite crear grupos mixtos. Esta opción es reciente, tiene tan solo meses y AWS ha anunciado que va a integrarse con los servicios de autoscaling tradicionales lo que conllevará que sea compatible con muchas herramientas y servicios actuales.

#### AWS Instance Scheduler

AWS Instance Scheduler[8] es una solución en el área de los Tuneups. Permite apagar y encender instancia o bases de datos en AWS. En comparación con Tuneup su funcionalidad es más reducida, no puede tratar con grupos de autoescalado tradicional de AWS y mucho menos con grupos Skytuneup, de los que no conoce. Tampoco es compatible con las soluciones Spot Fleet ni EC2 Fleet del propio AWS. La solución es una combinación de servicios de AWS que se puede desplegar automáticamente. La forma de configurar la solución es mediante la modificación manual de tablas, introduciendo identificadores de instancia y horas, no dispone de una interfaz gráfica por lo que a veces es difícil de usar por un cliente que no conoce detalles técnico sobre las tecnologías en la nube que componen el servicio.

#### Estado de Microsoft Azure

Los Low-Priority VM Scale-Sets de Azure están disponible para el público solo desde Marzo de 2018[9] y en régimen de Preview. Durante las fases iniciales de este trabajo Enimbos participo en una beta privada de esta funcionalidad durante la que se experimentó y empezó el desarrollo.

Anteriormente se ofertaba computación a precio reducido en Azure pero solo a trave de Azure Batch, que asume un modo de ejecución en bloque e implementa una cola de tareas que consumen maquinas de coste reducido pero que el usuario no puede gestionar directamente.

Quizás por el poco tiempo desde la implantación de las máquinas virtuales de baja prioridad en Azure aún no hay soluciones de proveedor para la gestión de grupos mixtos.

#### Estado de Google Cloud Platform

Google Cloud Platform ofrece su versión de recursos de computación a precio reducido en una modalidad de pago fijo similar a la de Azure. Su aplicación a las máquinas virtuales es Preemptible Instances. Estas instancias puede ser retiradas en cualquier momento, como un otros proveedores, y en cualquier cosa son retiradas a las 24 horas[10]. Google Cloud orienta su oferta de instancias de bajo coste a procesamiento en bloque con un foco en la computación científica y el aprendizaje automático y han expendido su oferta hacia recursos GPU de precio reducido para cubrir estas aplicaciones. Por otro lado Google no da una solución de proveedor para usar grupos mixtos en cargas de trabajo de otras clase.

### 2.2.2. Soluciones de terceros

#### Spotinst Elastigroup

Spotinst Elastigroup[1] es la solución de terceros más cercana a Skytuneup. Spotinst utiliza la misma combinación de soluciones que Skytuneup para posibilitar el uso de instancias de pago reducido.

Los puntos en los que Spotinst añade funcionalidad de la que Skytuneup carece son:

- Spotinst en la actualidad puede crear Elastigroups en AWS, Azure y Google Cloud Platform.
- Spotinst cuenta con una gran cantidad de integraciones con herramientas de terceros.
- Spotinst lleva integrado un sistema para hacer despliegues totalmente gestionados y sin perdida de servicio.
- Spotinst permite ejecutar componente con estado manteniendolo e intentando relanzar instancias con el último estado cuando sea posible. Esta funcionalidad engloba la funcionalidad de backup de Skytuneup.

Por otro lado Spotinst carece de las funcionalidad de Tuneup.

Este trabajo forma parte de un esfuerzo de Enimbos por competir con Spotinst.

[1] `https://spotinst.com/products/elastigroup/`

# Capítulo 3

# Diseño y desarrollo

En este capitulo se explica el diseño de Skytuneup, tanto el previo al trabajo como las nuevas decisiones que se han adoptado. También se aportan detalles de implementación.

## 3.1. Arquitectura de Skytuneup

Skytuneup tiene una arquitectura multicapa que pasamos a describir a continuación. Por cada capa explicaremos su función y daremos cuenta de su organización interna si tiene múltiples partes.

Empezaremos desde la capa más cercana al usuario que utiliza Skytuneup a través del portal web hasta los responsables últimos de realizar las acciones.

El propio Skytuneup se hospeda en la nube, más concretamente en AWS así que es importante no confundir que referencias a la nube hablan de la infraestructura de Skytuneup y cuales de la infraestructura que gestiona Skytuneup.

### 3.1.1. Frontend

El frontend, en el sentido de la parte de Skytuneup que se ejecuta del lado del usuario, es una aplicación web escrita usando el framework Angular en su versión moderna[1]. La navegación entre diferentes páginas del portal no requiere de recargar otra página nueva con todo su contenido sino que se realiza mediante llamadas a la API para pedir los datos necesarios y redibujar la parte necesaria de la aplicación. La aplicación web está formada por pequeños componentes que encapsulan una vista y un controlador ligero. El estado generalmente se encuentra en servicios separados que se pueden consultar desde los componentes y traen la información a través de la API según sea necesario.

### 3.1.2. Backend

A lo largo del backend se utilizan varias tecnologías y servicios de nube. Entre ellos el que más componentes soporta es ECS, el servicio para orquestación de contenedores de AWS. Un contenedor en un entorno virtualizado ligero, se parece en algunos aspectos a una máquina virtual pero en su implementación no es necesario emular hardware, simplemente el kernel del host separa un proceso o grupo de procesos de los demás en todos los aspectos: sistema de archivos, información sobre el entorno, redes, etc. Cada contenedor

[1] `https://angular.io/`

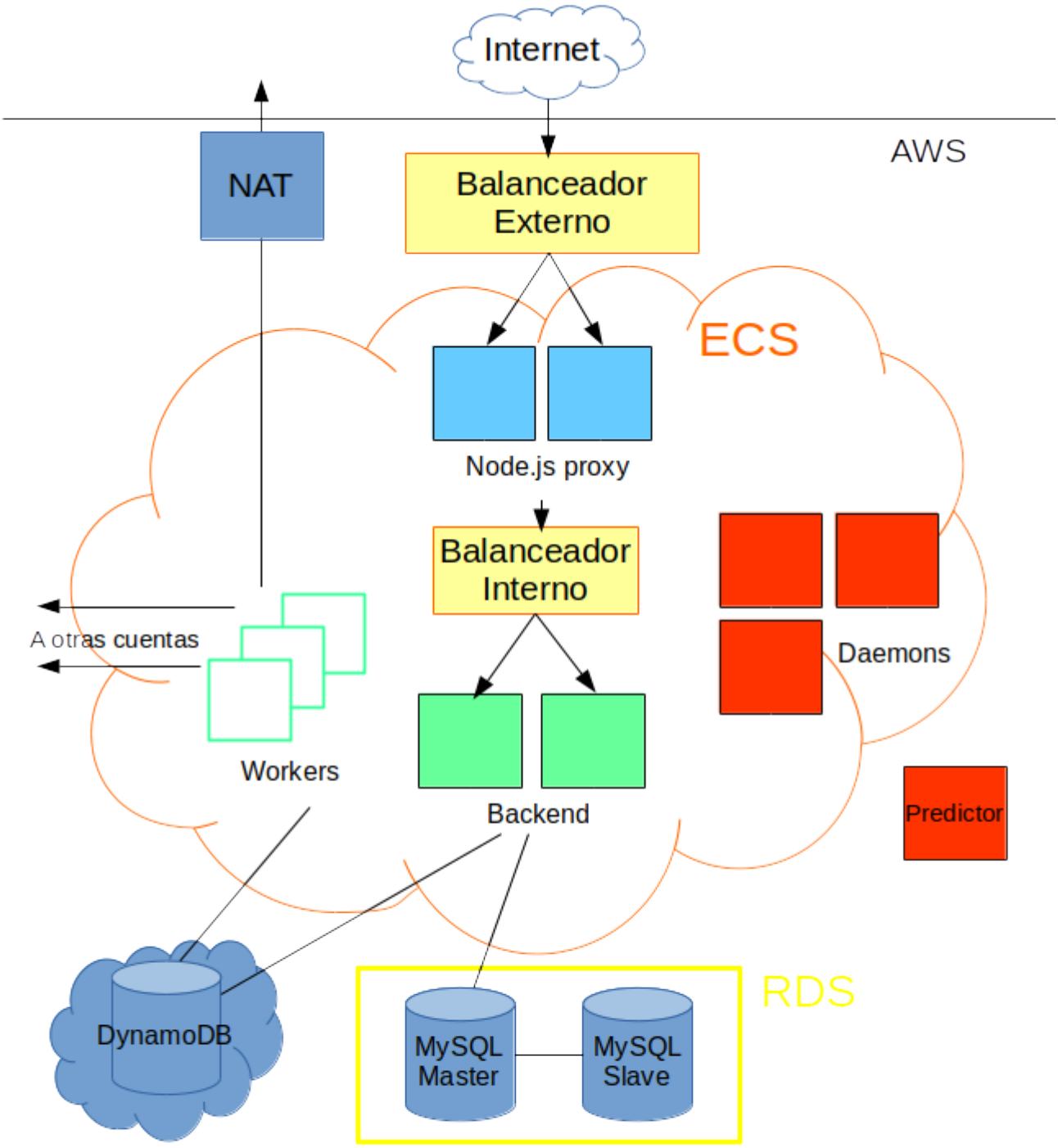


Figura 3.1: Diagrama de la arquitectura de Skytuneup

puede contener una aplicación o servicio y todas sus dependencias, lo que puede incluir todo el software de una distribución. De está forma es posible que los desarrolladores elijan tecnologías libremente sin entrar en conflicto que el mantenimiento de sistemas en la necesidad de dependencias concretas. Además, dado que es una virtualización ligera es posible para un host ejecutar muchos contenedores distintos.

Un orquestador de contenedores toma una serie de hosts, que pueden ser maquinas virtuales, y compone un cluster en el que reparte contenedores con las aplicaciones especificadas y en la cantidad deseada. Si un miembro del cluster desaparece el orquestador redistribuye los contenedores de la mejor forma posible. Es posible que los orquestadores realicen otra serie de tareas como comunicar los contenedores entre si pero en el caso de Skytuneup solo se usa ECS para repartir contenedores.

Como dato curioso, las maquinas que gestiona el propio cluster que ejecuta Skytuneup son gestionadas por Skytuneup. Para conseguir eso primero hubo que ejecutar Skytuneup en máquinas no gestionadas, crear un grupo Skyspot registrado con el mismo orquestador y dar de baja las VMs originales. De está forma se ahorran costes en la propia ejecución de Skytuneup y se da una oportunidad más de probar Skytuneup en un entorno real.

El backend es la capa que implementa la funcionalidad de Skytuneup. Está compuesto por varias partes pero la central es el servicio que provee los servicios de la API. Este servicio esta implementado en Python utilizando el framework Flask[2]. La API está documentada usando Swagger, que es una librería que genera una documentación interactiva que permite probar lo ejecutado, las definiciones de los parametros en Swagger se utilizan para validar el formato de la peticiones a la API, de esta forma la documentación y la implementación están en sincronía siempre.

Casi toda la API sigue el estilo REST. Este estilo se basa en el uso de los verbos de HTTP(GET, POST, PUT, DELETE) para representar varias acciones sobre un recurso o colección de recursos, generalmente operación de tipo CRUD. Las respuestas y parámetros de la API se codifican en JSON.

### Capa Node.js

De cara a Internet Skytuneup sirve el frontend y expone una API. La responsabilidad de exponer la API cae sobre un servidor ejecutando Node.js. Node.js[3] es una tecnología que permite ejecutar JavaScript en un entorno de servidor.

Esta capa no implementa ninguna de las funcionalidad de API y hace de proxy de los servicios de API. Al mismo tiempo es capaz de controlar y cuantificar los accesos a la API para cumplir criterios de seguridad y aprender sobre patrones de uso de Skytuneup.

### Workers

Para tareas costosas o que completan asíncronamente el backend crea contenedores dedicados a una tarea, estos contenedores solo viven tanto como dura la tarea, registran el éxito o fracaso y avisan al componente responsable de procesar el resultado en caso de que lo haya. Son los llamados workers o trabajadores y su razón de ser es liberar el servicio de API para que pueda responder a más peticiones. Dado que muchas interacciones con la API del proveedor de nube son asíncronas y se bloquean hasta terminar es beneficioso

[2] `http://flask.pocoo.org/`
[3] `https://nodejs.org`

convertir muchas en tareas asíncronas y procesarlas sin bloqueos y de una forma más generalizable. Un ejemplo de tareas es crear una VM on demand nueva.

#### Daemons

Gran parte de la funcionalidad de Skytuneup requiere actuar proactivamente y no solo como respuesta a peticiones. Para ese fin existen los demonios, en el sentido UNIX, contenedores o servidores que se ejecutan continuamente o con alta periodicidad para vigilar y realizar labores que son necesarias muy frecuentemente.

Algunos de estos demonios son:

- **Predictor de precios**
  Este demonio vigila los precios flotantes para todos los mercados relevantes, intenta predecir la evolución más probable de los precios y su rango en el corto plazo. Registra recomendaciones de precios de puja.

- **Vigilante del estado de los grupos Skyspot**
  Los grupos Skyspot están a merced de la retirada de instancias spot. Este demonio vigila si se ha producido una retirada o hay un aviso al respecto e intenta rebalancear el grupo. También puede intentar rebalancear el grupo si los precios o las predicciones de precios cambian. Todas esas decisiones de balanceo son procesadas por un motor de decisión que compara el estado actual con el estado deseado y aplica las preferencias recogidas en la configuración para dar un lista de acciones a tomar. Estas acciones se delegan en workers.

- **Vigilante de salud de instancias**
  El vigilante de salud de instancias no se centra solo en las retiradas de instancias spot sino en si las instancias, tanto spot como normales, se encuentran saludable según el hipervisor de AWS EC2 o según una prueba de salud definida por usuario. Si se produce un fallo aislado termina la instancia en cuestión y confía en el demonio vigilante de estado para restaurar la normalidad. También logea todos los problemas y si se producen fallos generalizados puede lanzar una alarma.

- **Programador de Tuneups y backups**
  Este demonio se encarga de la acciones periódicas programables en Skytuneup, esto es, las acciones Tuneup y los backups. Comprueba periódicamente la hora y la tabla de tareas y delega en workers las tareas cuando llega el momento.

### 3.1.3. Capa de persistencia

Skytuneup necesita almacenar datos de formas persistente. Se consigue persistir los datos gracias a una base de datos MySQL como servicio en AWS RDS. Y al uso de una base de datos no relacional como servicio DynamoDB.

Este área del diseño es la que más cambios ha sufrido en el integración con Azure. Más adelante se puede encontrar una discusión de los cambios cuya justificación explica el diseño de la capa de persistencia en el resto de Skytuneup.

## 3.2. Decisiones de diseño e implementación

En esta sección repasamos los elementos de la arquitectura tratando decisiones de diseño que se han tomado en ellos y como afectan a la implementación. Pero antes se expondrán decisiones de diseño que afectan a más de una capa del diseño.

### 3.2.1. Puntos de diseño transversales

El titulo de está subsección es puntos de diseño transversales. Se usa la palabra transversal es el sentido de que los contenidos de la subsección no caen dentro de ninguna capa especifica de la arquitectura y afectan a varias. Se trata de puntos que buscan armonizar Azure con la estructura de Skytuneup.

#### Estrategia para la construcción de grupos Skyspot

Para manejar un grupo Skyspot los datos mínimos que hace falta persistir son aquellos que describen el rango de estados deseados del grupo y aquellos que permiten descubrir el estado actual del grupo. Nótese que el propio estado real del grupo no es necesario guardarlo persistentemente, de hecho no es posible depender de un estado guardado porque el estado real puede cambiar en cualquier momento.

Teniendo en cuenta eso se ha optado por un modelo de datos como el de la figura 3.2.1. Los óvalos representan tipos de datos atómico u opacos de cara a Skyspot, los rectángulos son tipos de dato compuestos y las flechas que unen los datos representan referencias en el sentido de la flecha. Las referencias son a un solo objeto excepto las que tienen marcada su cardinalidad. Se han omitido campos como nombre y descripción que si que aparecen en la implementación real.

La información necesaria para recuperar el estado actual es muy pequeña. Unicamente es necesario mantener una lista de identificadores de Azure Scale-Sets(grupos de un tipo de VM y modalidad de pago) asociados con el grupo Skyspot. A partir de esa información es posible consultar a la API para saber cuantas VMs, de que tipo y en que estado se encuentran.

La descripción del estado del grupo Skyspot requiere más piezas, primero hace falta describir que tipos de instancia deseamos y cuanto valoramos cada una, después hay que especificar la capacidad deseada y la estrategia a seguir(un valor número que codifica el trade-off entre disponibilidad y ahorro). Esto deja tres valores, uno es la configuración para Azure de las instancias, esto es, todo los parámetros que Azure pide para crear una instancia pero que Skytuneup no necesita para tomar decisiones sobre el grupo Skyspot. Los otros dos valores son las capacidades máxima y mínima, su rol solo es importante cuando se aplica una política de autoescalado sobre el grupo Skyspot, una política puede modificar la capacidad deseada dependiendo de la situación, los valores mínimo y máximo sirven para acotar el rango de escalado automático.

Hay que señalar que las propias políticas de autoescalado tienen su modelo de datos separado que hace referencia a un objeto Skyspot.

En esta descripción del modelo de datos también se omite la descripción de las comprobaciones de salud personalizadas. Cuya implementación está pendiente.

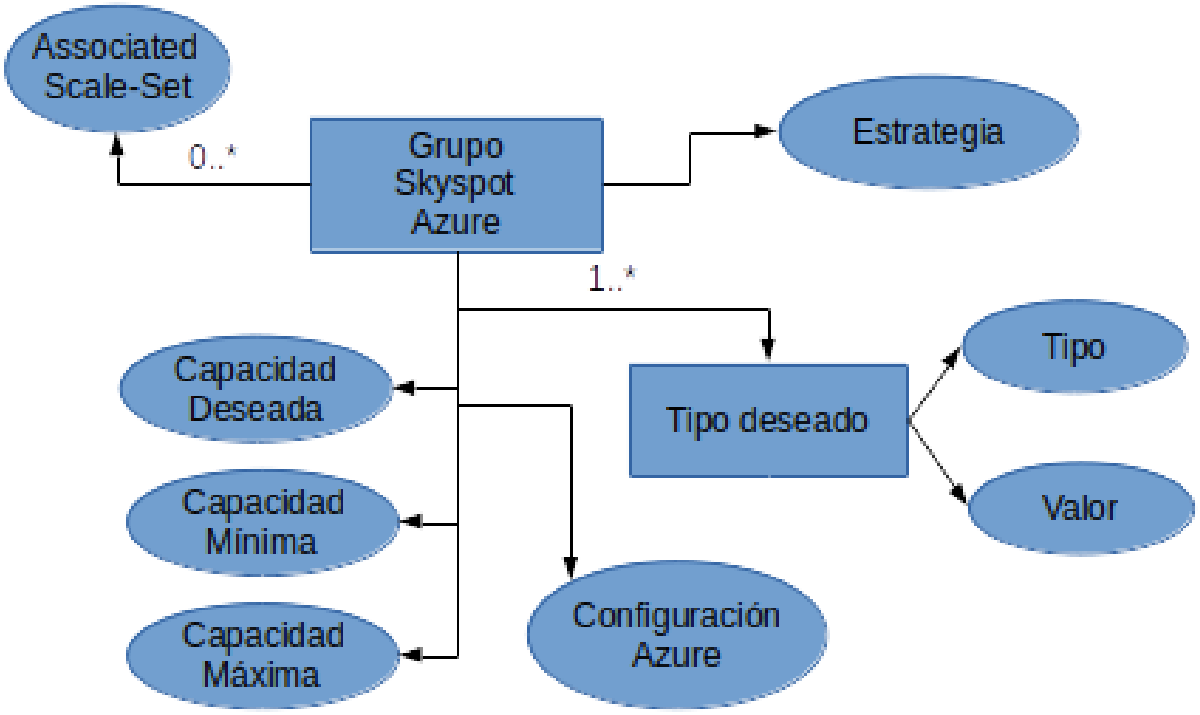


Figura 3.2: Diagrama de modelo de datos para un Skyspot en Azure

**Autenticación contra Azure**

Un aspecto que se tuvo que abordar pronto fue como autenticarse con Azure. No es posible realizar ninguna gestión en Azure sin antes resolver este punto. La solución de este aspecto no se parece casi nada a la solución existente para con AWS y se ha tenido que abordar desde cero.

Microsoft Azure ofrece varias formas de autenticarse progamáticamente:

- **Uso de credenciales token**
  La forma más sencilla es crear un llamado Service Principal en Azure, un usuario o rol con unos permisos definidos. Este Service Principal va asociado con unas credenciales que se pueden usar para llamar a la API de Azure o cargarse en librerias y SDKs para gestionar Azure. Hay pequeñas variantes se este metodo según como se carguen las credenciales pero son bastante equivalente entre ellos.

- **Managed Service Identity**
  Una instancia en Azure puede obtener una credenciales transitorias validas para gestionar su cuenta con los privilegios que se hayan otorgado a la instancia.

- **Autenticación con OAuth**
  Es posible pedir interactivamente a un usuario que otorgue permisos temporales y limitados a una aplicación mediante el protocolo OAuth[11]. El usuario solo ha de introducir sus credenciales seguramente en un popup controlado por Microsoft.

Explorando las opciones obviamente se descartó Managed Service Identity por imposible. De las opciones restantes se optó por un diseño que usase una autenticación OAuth, comoda, fácil y segura, para crear un Service Principal y guardar las credenciales. De esa forma es posible interactuar con la cuenta de Azure en cualquier momento futuro pero la experiencia del usuario es más cómoda. Actualmente el usuario solo tiene que introducir su id de subscripción, que es necesario para saber contra que cuenta intentar OAuth.

### 3.2.2. Frontend

En el frontend se han conservado la totalidad de decisiones de diseño. El framework para la nueva funcionalidad en Angular para integrarse fácilmente y poder reutilizar código relevante. Las facilidades para estructurar el código y el sistema de módulos hacer de Angular una opción natural para una aplicación grande. Otra razón es la posibilidad de implantar rendering en el lado del servidor. Una de las desventajas de frameworks como Angular basado en AJAX y en crear SPAs es que la carga inicial de la página es más lenta porque necesita de la ejecución de grandes cantidades de JavaScript(que inicialmente no está optimizado o transformado por el interprete o máquina virtual de JavaScript) y una o más peticiones AJAX para obtener la información inicial. El rendering en el lado del servidor es una forma sobreponerse a ese problema ejecutando la vista en el servidor, donde el JavaScript ya está optimizado y la información está disponible más rápidamente , y servir HTML en la primera petición para que el usuario tenga una aplicación usable sin esperas. Angular dispone de herramientas para realizar esto y que toda la aplicación use Angular con las mejores prácticas hace trivial que el código de la vista pueda utilizarse en navegador y servidor(Node.js) sin cambios importantes.

El lenguaje de diseño sigue siendo Material Design. Las razones son varias, es el lenguaje de diseño en uso en Skytuneup y por tanto usar otro daría lugar a un estilo

inconsistente, además Material es un lenguaje diseñado por Google, al igual que el framework Angular y gracias eso existe una librería con gran cantidad de componentes de gran acabado disponible e integrada perfectamente en Angular.

A pesar de utilizar las mismas tecnologías se plantearón cuestiones de diseño únicas derivadas del salto desde ser una herramienta para AWS a soportar varias nubes. Una cuenta de Skytuneup soporta varias cuentas de AWS y debería soportar varias cuentas de Azure. Primero se planteo la idea de tener 2 portales distintos a los que se podría navegar con un switch en la parte superior, en cada portal solo se listarían las cuentas de un proveedor de nube. Aunque está opción daba la mayor libertad a la hora de tomar decisiones nuevas para la integración de Azure no se adopto por razones de economia de implementación y por crear para el usuario una experiencia igual y mostrar que Skytuneup tiene paridad de funcionalidad en ambas nubes.

Aun así, en algunas partes de la interfaz habría que cambiar detalles y campos según que nube se utilizase. A la hora de decidir como hacerlo se planteo la pregunta de si bifurcar los componentes de la aplicación en sus versiones para AWS y sus versiones para Azure o mantener los componentes pero con una naturaleza dual según parametros del componente que indicasen que nube se estaba manejando. Al final se tomo una decisión mixta, bifurcando muchos de los componentes de menor nivel y preparando a los componentes de mayor nivel(los grandes paneles y areas generales) para instanciar y soportar componentes de cualquier nube según sea necesario. El criterio quedo de la siguente forma, si como componente todas las diferencias entre nubes puedes trasladarlas a subcomponentes entonces no necesitas ser bifurcado, en caso contrario sí. Para reducir el número de componentes con multiples versiones de refactorizarón algunos componentes moviendo partes a subcomponentes para solo duplicar lo estrictamente necesario.

### 3.2.3. Backend

En el backend se ha reevaluado la elección de tecnologías para la nueva integración. El resultado ha sido la decisión de usar las mismas tecnologías. Python con Flask en una combinación ideal para desarrollar un servidores de API que no necesita servir HTML. Además nos de la posibilidad de abordar una construcción modular de la API que si que se deseaba para permitir tomar una decisiones ligeramente distintas en la implementación mediante el uso de una construcción de Flask llamada blueprint[4].

La unica excepción en cuanto a elección de tecnologías es la adopción de la libreria ORM pony[5], esta libreria permite escribir las consultas en una sintaxis muy similar a manipular colecciones de Python con un estilo funcional y las transforma en el dialecto SQL que consuma el motor de base de datos usado, también permite crear relaciones en la base de datos a partir de la definición de los modelos casi como si fuese objetos normales de Python. Las queries SQL del resto de Skytuneup se construian a mano y las tablas se creaban y migraban por medio de script SQL escritos también a mano. Esto no suponia un problema porque gran parte de las relaciones estaban en DynamoDB y se usaba un ORM para generar las queries, aun así la creación de las relaciones necesitaba de trabajo manual no automátizado. La decisión de la adoptación de ponyORM deriva de decisiones de diseño para la capa de persistencia que se explican más adelante.

Anteriormente, las queries SQL estaban intercaladas con el resto de logica de aplicación dentro del manejo de peticiones. En el nuevo diseño se ha aplicado el siguente

[4] `http://flask.pocoo.org/docs/1.0/blueprints/`
[5] `https://ponyorm.com/`

criterio, se sigue podiendo hacer queries en medio del resto de la lógica únicamente si son operaciones CRUD simples, si se trata de operaciones compuestas como joins o de transacciones deben encapsularse en metodos del modelo, no en codigo que maneje el modelo. Inicialmente se penso en encapsular todas las operaciones sobre la base de datos en los modelos pero se decidio que substituir operaciones simples por llamadas a un modelo con los mismos parámetros no resta complejidad al código, hace más dificil leer el código(habría que visitar el modelo constantemente) y permite mentir en los nombres de los metodos o complicar un metodo en el futuro sin que sea aparente todos los sitios donde afecta y sin que haga falta en todos los usos de esas operación.

También se ha intentado avanzar en el uso de Swagger para la documentación de la interfaz de aplicación, anteriormente se especificaban los tipos de los parametros mediante definiciones indirectas de tal forma que a veces la versión ejecutable de la documentación no era capaz de asistir todo lo que pudiera en la creación de llamadas a la API porque no era capaz de inspeccionar los tipos de los parametros. También se ha adoptado la decisión de insertar la documentación Swagger en la docstrings de los métodos de Python de forma que la documentación este cerca del código que describe.

### 3.2.4. Capa de persistencia

En la capa de persistencia es donde se ha realizado uno de los cambios de diseño más significativos. La parte de Skytuneup para AWS utiliza en gran medida DynamoDB. La experiencia con DynamoDB nos ha demostrado algunos problemas. DynamoDB otorga a cada tabla una capacidad asignada medida en lecturas y escrituras por medida de tiempo, cuando uno es capaz de predicir su uso con facilidad y establece la capacidad asignada por encima pero no mucho de lo necesario puede conseguir ahorros en comparación a mantener una base de datos más tradicional en una máquina virtual. Por otro lado si los patrones de uso son más erraticos o variables es fácil, desaprovechar capacidad gran parte del tiempo y superar la capacidad en ciertos momentos. Para estos escenarios AWS recomienda usar un ajuste automático de la capacidad, el problema es que la capacidad siempre va a la zaga de la demanda y los aumentos de capacidad no son muy rápidos.

Por estas razones hemos decidido no usar DynamoDB en la nueva integración en favor de MySQL. Anteriormente ya se usaba MySQL para operaciones que requerian transaccionalidad pero en la nueva integración se usa para todas las necesidades de persistencia. La situación no llega a ser óptima porque buena parte del uso de la base de datos se hace en operaciones sencillas que no requiren de transacciones y quizas se lograrían aumentos en rendimiento, escalabilidad o economía al usar otro tipo de base de datos.

# Capítulo 4

# Pruebas y resultados

## 4.1. Tipos de pruebas

### 4.1.1. Pruebas unitarias

Las pruebas unitarias se limitan a las secciones del código con más lógica. El componente con más pruebas unitarias es el motor de decisión. Con ejemplos particulares se comprueba que el motor sabe distinguir estado que entra en el rango deseable de estado que se pueden mejorar, también se prueba que las acciones propuestas son las esperadas y el cambio de las preferencias en la configuración lleva a decisiones distintas en cosas borde según lo esperado.

### 4.1.2. Pruebas de integración

Se realizan pruebas de integración a nivel a la API que consisten en escribir y leer configuraciones. Las pruebas se crean y ejecutan con una herramienta llamada soapui[1].

## 4.2. Resultados del trabajo

La integración de Skytuneup es exitosa con carácter preliminar, aún se someterá a pruebas en situaciones reales antes de pasar a estar disponible generalmente y ser recomendada a los clientes de Enimbos que usan Azure.

El primero de los objetivos que se exponían, la paridad de funcionalidad respecto de AWS ha sido logrado casí completamente en el momento en que se redacta la versión final de esta memoria. El monitoreo de salud y la posibilidad de usar métricas arbitrarias en el escalado automático no están implementados todavía.

[1] `https://www.soapui.org/`

# Capítulo 5

# Conclusiones y trabajo futuro

En este último capitulo presentamos las conclusiones del trabajo y damos una idea del trabajo futuro tanto planeado como hipotético.

## 5.1. Conclusiones

En este trabajo se ha afrontado la integración de una herramienta para la gestión de núbe publica con otra nube distinta a la que originalmente estaba preparada para manejar.

Se plantearón dos objetivos: intentar replicar la funcionalidad de la forma más consistente y completa posible, y hacerlo de una forma mantenible y fácil de probar para que en el futuro se puedan abordar otros trabajos como este en nuevas direcciones.

Esos objetivos han informado las decisiones de diseño y, consecuentemente, la implementación.

Esa implementación se ha probado a varios niveles y tras muchos cambios ha pasado satisfactoriamente.

Ahora la implementación será completada y, volviendo a pasar las pruebas, tendrá la oportunidad de usarse en entornos reales.

Volviendo al principio, la nube es cada día más importante pero algunos aspectos y oportunidades siguen sin explotarse del todo. Este humilde trabajo ha intentado cerrar ese un poco un vacio muy concreto. Dado que los objetivos planteados se han alcanzado creo que la valoración del trabajo es positiva.

No obstante aún queda trabajo por hacer y cosas de que cerciorarse mejor para que este trabajo acabe siendo de provecho.

Un trabajo de diseño e implementación es solo un paso de un largo camino dentro y fuera de la disciplina de la ingenería de software para que una idea acabe siendo útil para alguien. Existe una cierta ayuda en considerar esa utilidad mientras se trabaja en un proyecto así. Cuando se toma una decisión pequeña pensando en todo lo que rodea la decisión las cosas caen en su sitio. Entre dos puntos una cuerda tensa solo puede tomar unos pocos caminos una cuerda no tensa puede enrollarse y dar vueltas sin sentido. Corresponde a los ingenieros no solo saber tensar la cuerda y evitar los obstaculos sino asesorar entre que puntos es más elegante tender la cuerda.

## 5.2. Trabajo futuro

En el futuro la integración sera probada en entornos reales en regimen de beta privada. El trabajo futuro incluirá realizar los cambios necesarios para solventar los posibles problemas que se encuentren.

A continuación se da una lista de posible tareas de trabajo futuro relativo a la integración de Azure con Skytuneup.

- Completar la implementación de el autoescalado con métricas arbitrarias.
- Completar la implementación de las comprobaciones de salud configurables.
- Añadir pruebas de unidad basadas en propiedades invariantes para el motor de decisión.
- Añadir pruebas end-to-end que prueben la interfaz de usuario.

# Glosario

- **AJAX**: Asynchronous JavaScript And XML
- **API**: Aplication Programming Interface
- **AWS**: Amazon Web Services
- **CPD**: Centro de Procesamiento de Datos
- **CPU**: Central Processing Unit
- **CRUD**: Create, Read, Update, Delete
- **EC2**: Elastic Compute Cloud
- **ECS**: Elastic Container Service
- **GCP**: Google Cloud Platform
- **GPU**: Graphics Processing Unit
- **HTML**: HyperText Markup Language
- **HTTP**: HyperText Transfer Protocol
- **IETF**: Internet Engineering Task Force
- **JSON**: JavaScript Object Notation
- **NAT**: Network Address Translation
- **ORM**: Object Relational Mapping
- **RDS**: Relational Database Service
- **RFC**: Request For Comments
- **REST**: REpresentational State Transfer
- **SDK**: Software Development Kit
- **S3**: Simple Storage Service
- **SLA**: Service Level Agreement
- **SPA**: Single Page App

- **SQL**: Structured Query Language
- **UI**: User Interface
- **UX**: User eXperience
- **VM**: Virtual Machine
- **XML**: eXtensible Markup Language

# Bibliografía


[1] *Gartner Says By 2020, a Corporate "No-Cloud" Policy Will Be as Rare as a "No-Internet" Policy Is Today*, https://www.gartner.com/newsroom/id/3354117, Consultado el 12 de Junio de 2018, 2016.

[2] D. M. Smith y E. Anderson, *Hype Cycle for Cloud Computing, 2017*, https://www.gartner.com/doc/3772110/hype-cycle-cloud-computing-, Consultado el 21 de Junio de 2018, ago. de 2017.

[3] Y. Brikman, *Terraform: Up and Running, Writing Infrastructure as Code*, 1.ª ed. O'Reilly, 2017, ISBN: 978-1491977088.

[4] D. Smith, L. Leong y R. Bala, *Magic Quadrant for Cloud Infrastructure as a Service, Worldwide*, https://www.gartner.com/doc/reprints?id=1-2G2O5FC&ct=150519&st=sb, Consultado el 21 de Junio de 2018, mayo de 2018.

[5] J. Barr, *Amazon EC2 Spot Fleet API – Manage Thousands of Spot Instances with one Request*, https://aws.amazon.com/blogs/aws/amazon-ec2-spot-fleet-api-manage-thousands-of-instances-with-one-request/, Consultado el 15 de Junio de 2018, mayo de 2015.

[6] ——, *New – Auto Scaling for EC2 Spot Fleets*, https://aws.amazon.com/blogs/aws/new-auto-scaling-for-ec2-spot-fleets/, Consultado el 15 de Junio de 2018, sep. de 2016.

[7] ——, *EC2 Fleet – Manage Thousands of On-Demand and Spot Instances with One Request*, https://aws.amazon.com/blogs/aws/ec2-fleet-manage-thousands-of-on-demand-and-spot-instances-with-one-request/, Consultado el 15 de Junio de 2018, mayo de 2018.

[8] A. Leeuwesteijn, M. ElZayet y R. Andreae, *AWS Instance Scheduler*, https://s3.amazonaws.com/solutions-reference/aws-instance-scheduler/latest/instance-scheduler.pdf, Consultado el 15 de Junio de 2018, 2018.

[9] *Public preview: Low-priority VMs on virtual machine scale sets*, https://azure.microsoft.com/en-us/updates/public-preview-low-priority-vms-on-vm-scale-sets/, mar. de 2018.

[10] *Preemptible VM Instances — Compute Engine Documentation*, https://cloud.google.com/compute/docs/instances/preemptible.

[11] D. Hardt, «The OAuth 2.0 Authorization Framework», RFC Editor, RFC 6749, oct. de 2012, http://www.rfc-editor.org/rfc/rfc6749.txt. dirección: http://www.rfc-editor.org/rfc/rfc6749.txt.